IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 16, 2021, accepted June 27, 2021, date of publication July 5, 2021, date of current version July 13, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3094980*

# Fault Tolerance Framework for Composite Web Services

**NIMRA MEMON[1], MUHAMMAD SALEEM VIGHIO[2], YAR MUHAMMAD[3], (Senior Member, IEEE), AND ZAHID HUSSAIN ABRO[1]**

[1]Department of Information Technology, Quaid-e-Awam University of Engineering, Science, and Technology, Nawabshah 67480, Pakistan
[2]Department of Computer Science, Quaid-e-Awam University of Engineering, Science and Technology, Nawabshah 67480, Pakistan
[3]Department of Computing and Games, School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough TS1 3BX, U.K.

Corresponding author: Muhammad Saleem Vighio (saleem.vighio@quest.edu.pk)

**ABSTRACT** A composite Web service combines multiple, logically interrelated services for creating more common services meeting complex requirements from users. The services participating in a composition coordinate the actions of distributed activity using Web services protocols to reach consistent agreement on the outcome of joint operation. However, as services run over unreliable protocols, there is a great chance that services fail due to the failure of protocols. However, current protocol standards provide fault-tolerance but are limited to backward recovery using expensive compensation and roll-back strategies. This paper gives an extension of the existing Web services business activity (WS-BA) protocol to deal with failures using forward recovery approach. A set of common failure types affecting the execution of component services is identified, and recovery solutions for each identified failure are also presented. The fault-handling extension of the WS-BA protocol implements recovery solutions for each of the identified failures to handle failures at runtime. Another important aspect about which the WS-BA protocol specification is unclear is reaching and notifying consistent outcome on the completion of joint work. This study extends the WS-BA protocol to notify consistent outcome reached by all participating services. The implementation and testing of the framework are performed using the model-checking and verification tool UPPAAL. A well-known application example supports the study. The key properties of the framework, like the execution of corresponding recovery actions in cases of failures and reaching a consistent agreement on the outcome of joint operation, are verified.

**INDEX TERMS** Web services, fault handling, forward recovery, model-checking, transaction protocols.

## I. INTRODUCTION

A Web service is a software application that encapsulates logic and performs a specific task using the Internet [1]. With the growing use of the Internet, Web services have gained much popularity, and nowadays, most of the service demands from users are being answered through the Web [2]–[3]. In general, one single service has relatively simple functionality, and in many cases, a single service on its own is not sufficient to perform a complex task independently; for example, a travel reservation task may require booking of an air-ticket, a hotel room, and a taxi to fulfil a reservation request. In such a case, multiple services are combined into a single service to perform that task jointly and in an agreed-upon manner [4]. The process of aggregating multiple services into a single service is called composition and is facilitated using service-oriented architecture (SOA) standards. Among these standards, Web services coordination and agreement (WS-C&A) protocols allow multiple services to coordinate the actions of activities that require to reach a consistent and agreed-upon outcome [5]–[7]. However, like other communication protocols, Web services protocols also suffer from errors and failures during execution [8]–[9]. In addition to that, current protocol standards are limited to backward recovery and handle failures using strict roll-back and compensation strategies [10]–[11]. This makes current protocol standards very expensive and time-consuming processes, specifically for applications that run for longer durations. Due to the extensive use of Web services in IoT (Internet of Things), AI (Artificial Intelligence), and other sensitive and mission-critical applications, Web services are required to be highly reliable even in cases of failures. This paper provides a fault-tolerant framework for composite Web services by implementing fault-tolerance in Web services

The associate editor coordinating the review of this manuscript and approving it for publication was Zhangbing Zhou.

business activity (WS-BA) protocol based on forward recovery approach. The WS-BA protocol is designed to allow independent services to join in common activities which run for a longer duration and require to reach a consistent outcome [7]. However, in its current settings, WS-BA protocol deals with failures using compensating actions in backward recovery fashion; that is, in case of a failure of a participant service, the effects of previously completed tasks are undone (or reverted). This paper provides an extension of the WS-BA protocol to deal with failures using forward recovery approach. In this approach, when a failure occurs during the execution of a service, its corresponding recovery action is invoked to recover from that failure rather than starting the operation all over again or reverting the effects of previously completed tasks. Compensation or strict roll-back is a severe issue with services that run for longer durations, usually lasting for days or weeks [12]. To conduct this study, we identify common failure types affecting the execution of composed Web services. Moreover, recovery actions for each of the identified failures are also identified. The fault-handling extension of the WS-BA protocol implements proposed recovery solutions against each of the specified failures types to diagnose and handle failures at runtime. To simplify the development process and satisfy varying application requirements, exception handling logic is separately implemented from the actual business logic. Another key issue tackled in this study is to reach and notify consistent outcome on the completion of joint work about which the BA protocol specification is not precise. The implementation and testing of the framework are performed using the model-checking and verification tool UPPAAL [13]. A well-known travel reservation scenario supports the study. The key properties of the system like execution of corresponding recovery solutions in cases of failures and reaching a consistent outcome on the completion of joint operation, are verified.

The remainder of the paper is organized as follows: Section II gives a brief review of related work. An overview of the WS-BA protocol is presented in section III. Following that, section IV presents proposed extensions to the existing WS-BA protocol and gives details of the proposed fault-tolerant framework. Failures common to Web services and recovery solutions are provided in Section V. Section VI presents supporting application example. Implementation of the framework and verification results are provided in Section VII. Finally, Section VIII presents the conclusion of the work and gives directions for future work.

## II. RELATED WORK

Web services have gained much popularity in recent years because of their extensive use in IoT, mobile, AI, and cloud applications. Gartner, Inc., the world's leading research and advisory company, has identified Web services as the most prominent area in its ''Top 10 Strategic Technology Trends for 2018'' when integrated with other technologies like AI, mobile, and cloud [14]. Forbes, another world-leading company, predicted that by 2020, 80% of companies would provide services to their customers enriched with the latest tools and technologies [15]. On the one hand, the use of Web services is on the high rise; on the other hand, it is becoming a challenging task for the researchers to provide and maintain the reliability of Web applications [16]. The failure of Web services is a real issue that occurs due to many reasons like the failure of a resource, error in underlying service logic, failure of communication protocols, and so on [17]–[18]. The failure of services may result in services downtime or complete failure leading to a situation ranging from simple inconvenience to a significant financial or monetary loss, or even the loss of human lives. Due to the increasing use of Web services in important, sensitive, and mission-critical applications, Web services are required to be highly reliable even in cases of failures [19]–[21]. However, as services run over unreliable protocols and communicate beyond organizational boundaries in heterogeneous environments, Web services are vulnerable to a wide variety of failures than traditional software. Efforts to provide fault-free services at the application level have received much attention. However, due to the complexity of the problem, little emphasis has been given to providing fault-tolerance at the level of protocols, especially at the level of Web services protocols [22]. Moreover, current protocol standards are limited to backward recovery and deal with failures using compensation and roll-back strategies, which is a time-consuming and expensive process, especially when services are designed to run for longer durations [23]–[24]. To provide fault-tolerance in Web services protocols, Yang and Liu [25] propose an extension of the existing WS-BA protocol by incorporating flexible compensation to deal with failures. The flexible compensation is used to satisfy various requirements from different applications as the existing standard is too fixed to deal with varying application requirements. In a similar approach, Schäfer *et al.* [26] provide an extension of WS-BA protocol which allows replacement of failed services with alternative services using compensation mechanisms. Some researchers employed exception handling strategies to realize the backward recovery; for example, Liu *et al.* [9] present a framework named FACTS for fault-tolerance of transactional composite services. The framework incorporates exception handling and transaction techniques to improve the fault tolerance of composite Web services. Initially, a set of high-level exception handling strategies are identified, and after that, a specification module is designed to help service designers build the correct logic for fault handling. Finally, a module is devised to automatically implement fault-handling logic in WS-BPEL. In another effort, Cardinale *et al.* [27] propose a framework for fault-tolerant execution of transactional composite services. Relying on compensation protocol, the framework deals with failures using replacement strategy in forward recovery fashion; that is, when a component service encounters a failure, that service is replaced with an alternate service having equivalent functionality. However, the paper lacks information on which failures-types and which recovery actions can be considered. Furthermore,

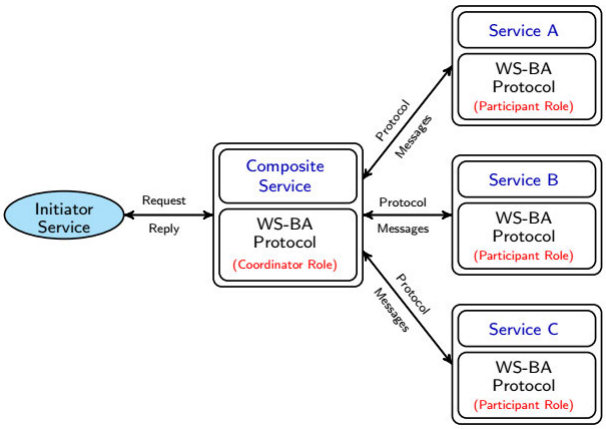


**FIGURE 1.** Coordinator and participant roles of WS-BA protocol.

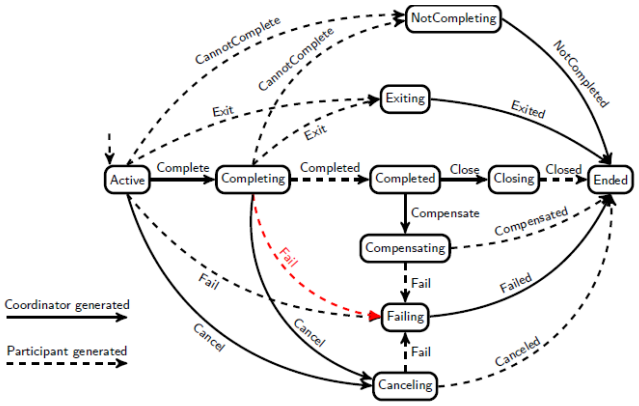


**FIGURE 2.** WS-BA protocol state transition diagram, adapted from [7].

the separation concepts have also been overlooked; for e.g., if the same framework is to be used with different applications with varying requirements, then how it would fit in that environment is unclear. In our case, we implement fault-handler as a separate process (from the normal business process) which can be used with different application examples. Zeng *et al.* [28] present a policy-driven approach to exception handling for composite Web services. In this approach, exception handling logic is separately implemented than the normal business logic. The authors argue that the separation of concerns significantly reduces the process development time and gives the flexibility to be used with different application environments. In all the above works, the main property of reaching consistent agreement on the outcome of the joint operation is missed. In this paper, we verify that the failures of participant services are dealt by executing corresponding recovery actions automatically and verify the important property of reaching and notifying the consistent outcome of the joint operation.

## III. WEB SERVICES BUSINESS ACTIVITY PROTOCOL

Built on the top of WS-Coordination [5], WS-BA protocol coordinates the actions of long-running distributed applications, which require reaching consistent outcome [7]. As shown in Fig. 1, the protocol defines two roles for the exchange of messages between the participating services: Coordinator and Participant. A composite service registers with the Coordinator role of the protocol, whereas component services register with the Participant role of the protocol for communication. As shown in the abstract diagram of the protocol in Fig. 2, when a Participant service encounters a failure or is not able to complete its work, it sends a **Fail** message to the Coordinator service and changes its state from **Completing** to **Failing**. The Coordinator service, upon receipt of the **Fail** message, sends **Compensate** message to all other participant services to undo their completed work. This is a severe issue with the current settings of WS-BA protocol that it does not provide any remedy to deal with failures rather than to compensate in backward recovery fashion. The compensation action results in the loss of precious work that has already been completed in the long-running environment. Moreover, the protocol specification is also not precise on the overall outcome of the joint operation, whether it should be committed or aborted. We provide extensions to the WS-BA protocol by implementing a fault-handler to deal with failures in forward recovery fashion. Furthermore, the key property of reaching a consistent decision on the completion of joint operation has also been considered. The details of the proposed extensions of WS-BA protocols are provided in the section to follow.

## IV. FAULT-TOLERANCE FRAMEWORK

As discussed previously, in a composite environment, multiple services register for Web services protocols to participate in activities whose completion requires consistent outcome. Fig. 3 shows a scenario in which WS-BA protocol is extended with a Fault-handler. The Fault-handler implements a pool of recovery actions to deal with common failure-types occurring in participant Web services. If a participating service encounters a failure, the type of the failure is communicated to the Fault-handler using the protocol instance of the participant service. The Fault-handler, in turn identifies the type of failure and invokes corresponding recovery action from the pool to resolve that failure, see Fig. 4 and 5. The resolution of failure is communicated to the protocol instance of the participating service so that the remaining computation can be completed. The details of each of the components of the Fault-handler are given below.

### A. COMPONENTS OF FAULT-HANDLER

As shown in Fig. 5, the Fault-handler deals with failures in the following two phases:

#### 1) FAULT DIAGNOSIS

When a participant service encounters a failure, the type of failure is communicated to the Fault-handler using the protocol instance of the service with which it registers. In its first phase, the Fault-handler diagnosis and identifies the type

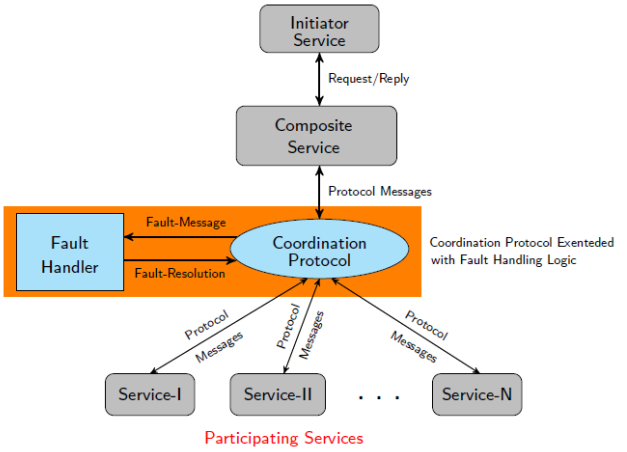


**FIGURE 3.** Fault-tolerance framework.

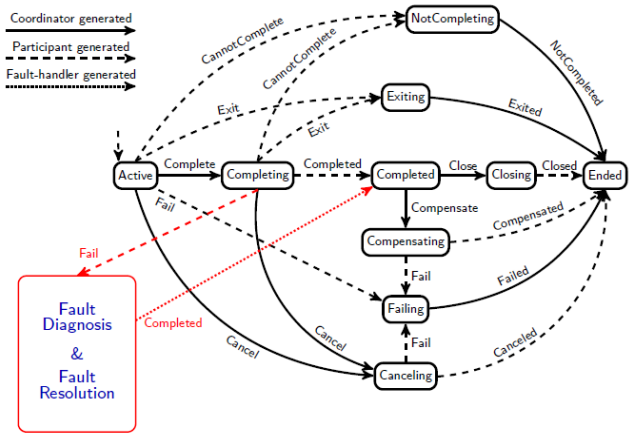


**FIGURE 4.** Fault-handler extension of WS-BA protocol.

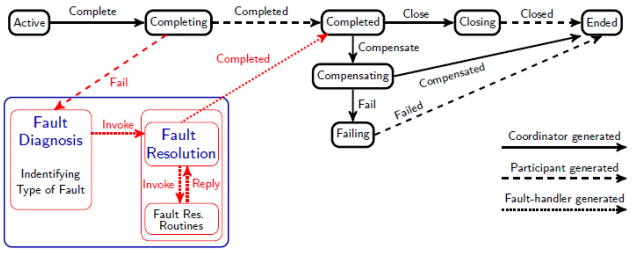


**FIGURE 5.** Fault-handler components.

of failure that has occurred at participant Web service so that its corresponding recovery action can be executed to recover from that failure.

#### 2) FAULT RESOLUTION

In the next phase, the corresponding recovery action for the identified failure is executed. For each of the failure-type, we implement a corresponding recovery routine. In cases when a single recovery action is unable to resolve a failure, a combination of different recovery actions is executed (see, Sec. V). After the failure has been resolved, the protocol completes the remaining execution in the usual order as defined in the protocol specification.

## V. WEB SERVICES FAILURES AND RECOVERY STRATEGIES

### A. WEB SERVICES FAILURES

Web services, like other software components, suffer from errors and failures from development to execution. Additionally, as Web services run over unreliable protocols under heterogeneous environment they are more susceptible to failures than their traditional counterparts [29]–[31]. Among all fault-types Web services suffer from, we consider participant services failures which occur when services are invoked through Web services protocols. Participant fault-types are classified into system (or physical) faults, inconsistent (or logical) faults, and interaction faults.

- System faults include all fault classes which affect hardware. System faults occur due to the failure of hardware (hosting server crash), software (operating system, database, error, or malicious attack), or communication infrastructure (network). In all the above cases, services become unavailable. For example, flight and hotel services may be unavailable due to hardware, software, or network failures.
- Inconsistent faults occur when the interface or ontology of the service is changed (or updated), but the users are unaware of corresponding changes. In some other cases, the service interface is changed, but the process (logic) is not updated accordingly. For example, in the case of flight service, a user tries to book two air tickets, but only one (or no) ticket is available at that time.
- Interaction faults are all operational or external level faults that arise when services are actually executed. These fault types are further classified into QoS (Quality of Service) and Time-out faults.
  - QoS exceptions are raised when a partner service completes, but execution results do not adhere to the predefined values. For example, the expected operation completion time is 12 seconds, but the actual operation took 20 seconds to complete.
  - Time-out faults occur when the service is overloaded to process too many requests simultaneously. For example, too many requests for grabbing a cheap ticket may overload the booking service. This may result in excessive delays (time-outs) at the requester's end or even in the unavailability of the service.

### B. RECOVERY STRATEGIES

A recovery solution lets the service operate correctly even in a case of failure. Specific to our application requirements, Table 1 gives details of the most common recovery solutions for Web services [8], [9]. The recovery solutions provided

**TABLE 1.** Recovery strategies.

| Strategy | Description | Example |
|---|---|---|
| **Skip(s)** | The action specifies that service is not needed to execute due to such reasons as to comply with cost and time constraints. Only the execution of those services is skipped which do not affect the primary goal of the overall computation. Generally, successor services of the faulty service are skipped to execute in order to meet the QoS aspect of the overall computation. | `skip (getSalesInfo)` meaning to skip `getSalesInfo` service. |
| **Retry(S,n)** | It re-executes the faulty service to a specific number of times or until the service completes successfully. This action is used to recover from temporary faults caused by hardware, software, or network. | `Retry(bookFlight,3)` meaning to invoke `bookFlight` service to a maximum three times. |
| **RetryUntil(s,n,duration)** | With addition of time-based re-invocation of the faulty service, this strategy is an extension of the 'Retry' strategy. Under this action, a faulty service is re-invoked to a specific number of times, with each re-invocation constrained to a particular time-stamp. | `RetryUntil (bookFlight,3,10)` meaning to re-invoke bookFlight service to a maximum three times with each re-invocation to take place after ten time-stamps. |
| **Wait (S, deadline)** | This strategy delays the execution of a service to a specified duration of time. | `Wait (bookFlight,8:00)` meaning to invoke `bookFlight` service at 8 o'clock. |
| **Alternate (s1,s2)** | When a particular service fails, its functionally-equivalent service is called to perform the task. Alternative action invokes different services instead of the same service. | `Alternate(bookFlight,bookTrain)` meaning to invoke `bookTrain` service as an alternate of `bookFlight` service. |

**TABLE 2.** Failures and corresponding recovery actions.

| Fault Type | | Recovery Action |
|---|---|---|
| System | | Wait, Retry, RetryUntil, Alternate |
| Inconsistent | | Wait, Retry, RetryUntil, Alternate |
| Interaction | QoS | Ignore, Skip |
| | Time-out | Skip, Wait, Retry, RetryUntil, Alternate |

in Table 1 can be used individually or in combination with other recovery solutions to recover from a specific failure. Table 2 shows which recovery solutions can be used against which types of failures.

Notably, the order in which recovery actions execute is essential from the implementation point of view. For our implementation, the execution order of proposed recovery actions is given in Fig. 6. This means that, `Wait` strategy is followed by the `Retry` strategy which in turn is followed by the `RetryUnitl` and `Alternate` strategies. Furthermore, we also consider that `Skip` action cannot be used in combination with any other strategies; that is, once the execution of any service is skipped, it cannot be re-invoked again.

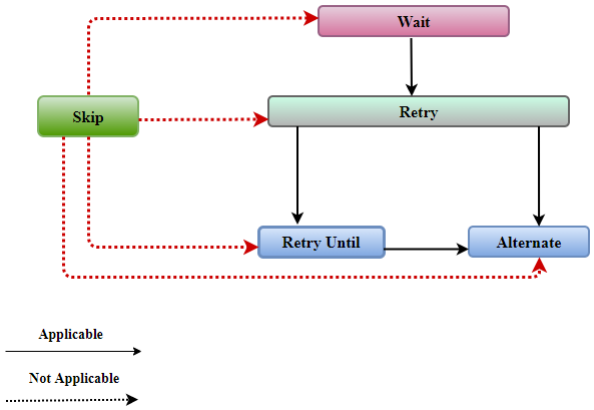


**FIGURE 6.** Recovery solutions hierarchy.

## VI. TRAVEL RESERVATION PROCESS

Implementation of the framework is supported by a well-known travel reservation example [34]. As shown in Fig. 7, the Travel agent service is implemented as a composite service, and all other services are implemented as component (or participant) services. All services register with a separate instance of the WS-BA protocol for communication and reaching a consistent outcome. Initially, the Client service sends a reservation request to the Travel agent (TA) service. The TA service, in turn, sends the request to the Airline, Attraction, and Hotel services. The Compute-Distance service computes the distance between the Attraction and Hotel services, and if the distance between the two services is greater than 5km, the Car service is invoked; else, Bike service is invoked. The Shop service is implemented as an optional service to test the QoS aspect of the overall reservation task. The reservation task is considered a joint task participated in by all services to reach a consistent and agreed-upon outcome.

## VII. FRAMEWORK MODELLING AND VERIFICATION

### A. UPPAAL MODELLING

The framework is implemented using the model-checking and verification tool UPPAAL [13]. For the implementation purpose, the following assumptions are set:

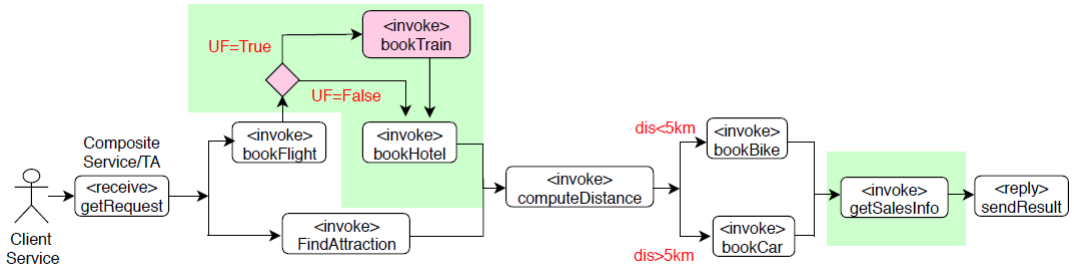


**FIGURE 7.** Travel arrangement process, adapted from [36].

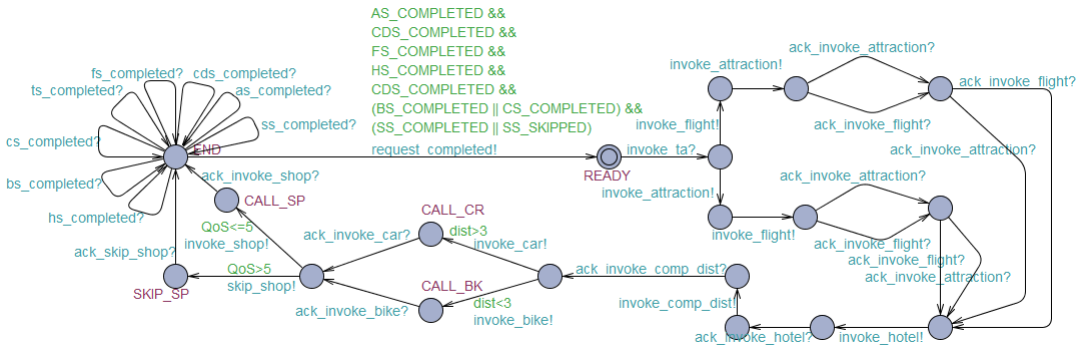


**FIGURE 8.** Travel agent service.

- All the services shown in Fig. 7, register and communicate using a separate instance of the WS-BA protocol.
- Travel agent service is considered the composite service and registers using the protocol's coordinator role for communication with component services.
- All other services register and communicate with composite service using the Participant role of the protocol.
- Failures are introduced in the Travel agent, Flight, and Hotel services such that Flight service may encounter service unavailability, seat unavailability, and time-out failures. Hotel service may suffer from service unavailability and time-out failures, whereas the Travel agent service implements QoS requirements.
- Shop service is considered an optional service and is used to test the QoS requirement of the overall reservation process.

Furthermore, to recover from failures following assumptions are made:

- Service unavailability and time-out failures are dealt using `Wait`, `Retry`, `RetryUntil`, and `Alternate` strategies.
- Seats Unavailability is handled using `Alternate` strategy and,
- QoS requirement is met using `Skip` strategy.

Rest of the services Client, Attraction, Compute-Distance, Car, Bike, and Shop, are considered fault-free services. Table 3 shows failure-types and corresponding recovery solutions considered for implementation.

### 1) TRAVEL AGENT SERVICE

The UPPAAL model of the underlying logic of the Travel agent service is shown in Fig. 8.

- Travel agent service is implemented as a composite service and is responsible for invoking all participant services to fulfil the reservation request.
- The service also implements the QoS requirement of the reservation task, i.e., if the overall task takes longer to complete, then the execution of optional Shop service is skipped. QoS is implemented as a global variable of type `integer` initialized to `0`. With each retry of Flight and Hotel services, it is incremented by a one time-unit, and if the QoS value of the overall reservation task is greater than `5` time units, execution of optional Shop service is skipped.
- After all the services complete execution, the Travel agent notifies the decision to the Client service.

### 2) FLIGHT SERVICE

Given the UPPAAAL model in Fig. 9, Flight service may encounter service unavailability, seat unavailability, and time-out failures.

- Service unavailability is dealt with using `Retry` recovery action.
- Each retry is constrained (time-out period) to take place after `3` timestamps implemented using the `wait` variable in the Fault-handler (see, Fig. 11).
- The maximum number of retries is set to `3` tries.

**TABLE 3.** Travel reservation process: Failure and recovery strategies considerations.

| Fault Type | Name of Service | | | Recovery Strategy |
|---|---|---|---|---|
| | Travel Agent | Flight | Hotel | |
| System | — | Flight Unavailable | Hotel Unavailable | Wait, Retry, RetryUntil |
| Inconsistent | — | Seats Unavailable | — | Alternate |
| Interaction | QoS | — | — | Skip |
| | — | Service Timeout | Service Timeout | Wait, Retry, RetryUntil |

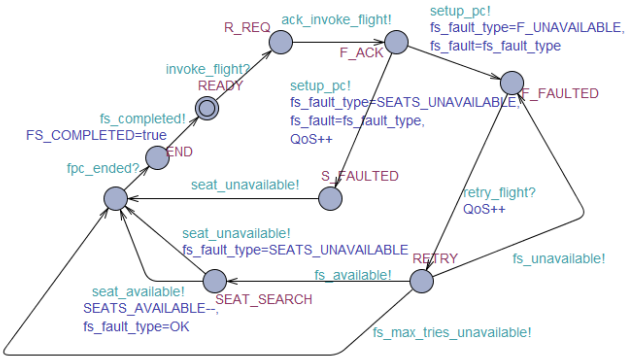


**FIGURE 9.** Flight service.

- If the maximum number of retries fails, the service is considered to have a dormant fault, and the Fault-handler calls the alternate service (Train service in our case).
- In case if retry is successful, one of the following two conditions happen:
  - Either seat may be available and reserved for the Client (`SEATS_AVAILABLE--`), or,
  - Seat may not be available to represent inconsistent failure. The unavailability of a seat is communicated to the Fault-handler using the `seat_unavailable!` action, which in turn executes the `Alternate` strategy.

### 3) HOTEL SERVICE

The UPPAAL model of the Hotel service is shown in Fig. 10. The Hotel service encounters either service unavailability or time-out failures.

- Service unavailability is handled using `Retry` recovery solution.
- Each retry is constrained (time-out period) to take place after `3` timestamps implemented using the `wait` variable in the Fault-handler (see, Fig. 11)
- The maximum number of retries is set to `3` tries.
- It is assumed that after the maximum number of retries, service resolves temporary fault and makes a room reservation (`ROOMS_AVAILABLE--`).

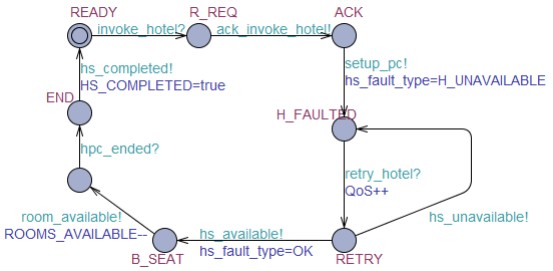


**FIGURE 10.** Hotel service.

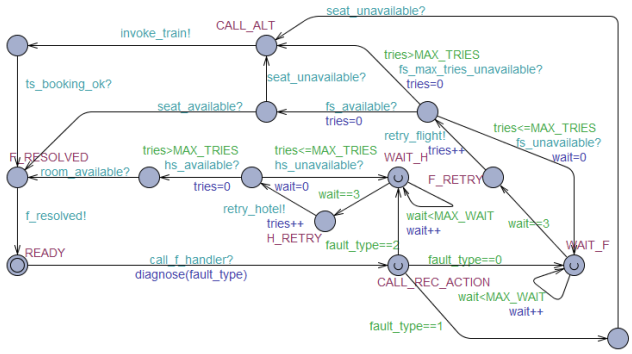


**FIGURE 11.** Fault handler service.

### 4) FAULT-HANDLER SERVICE

Fault-handler is responsible for identifying the type of failure and invoking a corresponding recovery solution at the failed service. As shown in the UPPAAL model of the Fault-handler in Fig. 11, the identification of a failure is implemented using the function `diagnose(fault_type)`, the code of which is shown below:

```
void diagnose(FaultType ftype)
{ int n; int f_type[ALL_FAULTS];
    for(n=0; n<ALL_FAULTS; n++)
        {if (ftype==f_type[n])
           {fault_type=ftype;} }
}
```

After receiving the failure information, the Fault-handler identifies the type of failure from the pool of failures

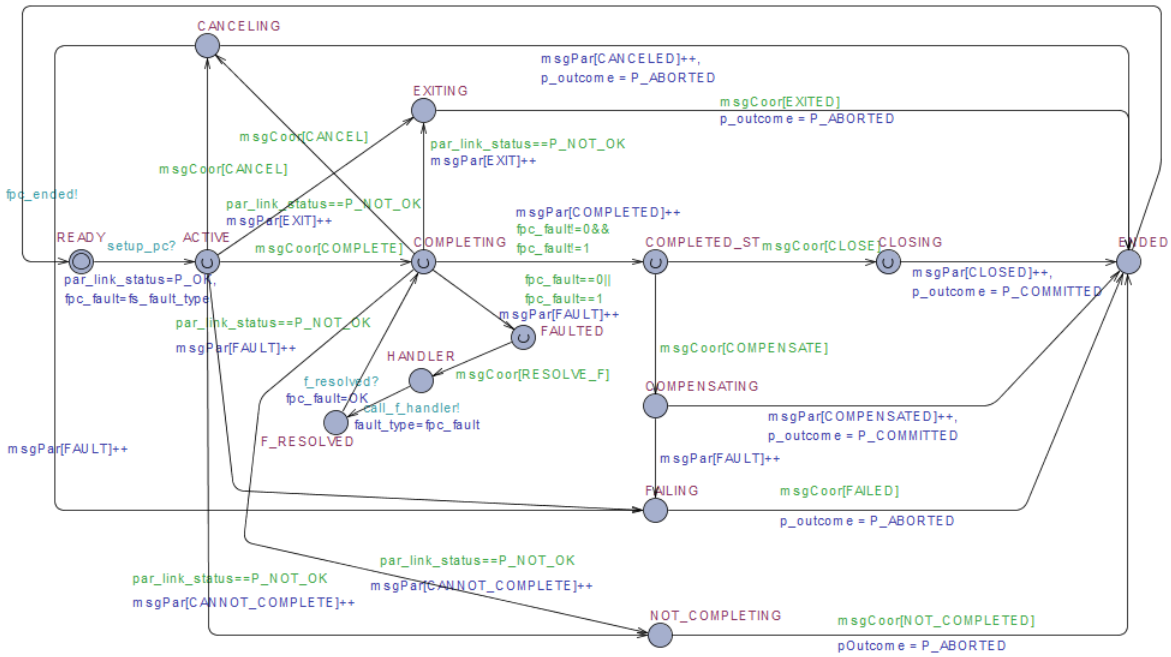


**FIGURE 12.** Extended WS-BA protocol.

implemented as an array (`f_type[ALL_FAULTS]`). If the received failure type matches a failure in the pool, the corresponding recovery action is invoked. As shown in Fig. 11, fault types `0`, `1` and `2` represent flight unavailable, seats unavailable, and hotel unavailable faults, respectively. In case if Flight and Hotel services are unavailable, `Retry` recovery action is called at the participant service. Each retry is constrained to take place after `3` time-units implemented using the `wait` variable of type `int` initialized to `0`. The maximum number of retries (`MAX_TRIES`) is constrained to `3` retries. In the case of Flight service, after the maximum number of retries expires (`tries>MAX_TRIES`), the service is considered to have a dormant fault, and the Fault-handler calls the alternate service (Train service in our case). For the Hotel service, after the maximum number of retries, the service resolves the unavailable service fault, and the room is reserved for the client. After the failures are resolved, Fault-handler notifies it to the protocol instance of the corresponding service so that the remaining execution can be completed.

### 5) WS-BA PROTOCOL

As mentioned earlier, all services participating in the reservation process communicate using an instance of the WS-BA protocol. Composite service (Travel agent in our case) communicates using the Coordinator role of the protocol, and all other services communicate using the Participant role of the protocol. The UPPAAL model of the Participant role of the protocol is shown in Fig. 12. When a participating service confronts to a fault, corresponding protocol instance records that failure-type using the instruction `fpc_fault = fs_fault_type`. The protocol instance then calls the Fault-hander (`call_f_handler!`) for the resolution of failure by sending it the fault information (`fault_type = fpc_fault`). After the fault has been resolved, the Fault-handler informs the protocol instance by sending it `f_resolved` message to indicate that the fault has been resolved. Another important aspect implemented in the protocol is to reach and notify the common agreement on the outcome of the joint work about which the WS-BA protocol specification is imprecise. This is implemented using a structure variable `pOutcome` as shown below:

```
typedef int[0,2] OutcomeP;
const OutcomeP P_ABORTED = 1;
const OutcomeP P_COMMITTED = 2;
OutcomeP pOutcome;
```

We have considered two types of outcomes: `P_COMMITTED` referring to the successful completion of the work and `P_ABORTED` when the work does not complete successfully. This is an essential property of the WS-BA protocol by which the composed services reach a joint decision after the operation has been completed.

### 6) ATTRACTION, COMPUTE-DISTANCE, CAR, BIKE, SHOP AND TRAIN SERVICE

Attraction, Compute-Distance, Car, Bike, Shop, and Train services are considered non-faulty services and contribute to completing common reservation tasks. **Attraction** service is a simple service that interacts with the Travel agent service. This service registers with a separate instance of WS-BA protocol to participate in the common reservation

**TABLE 4.** Verification results.

| UPPAAL Property | Status | Description |
|---|---|---|
| **Flight Service** | | |
| A[] f_handler.WAIT_F and f_handler.wait>3 | Not Satisfied | There is never the case that the Fault-handler waits for more than 3 time units before calling "Retry" action. This means that a "Retry" action is called each after 3 time units. |
| A[] f_handler.tries>3 imply not(f_handler.F_RETRY) | Satisfied | It is always the case that after the maximum number of retries, Fault-handler never calls the "Retry" action for the Flight service. |
| E<> erratic_flight.F_FAULTED and erratic_flight.fs_fault==F_UNAVAILABLE imply erratic_flight.RETRY | Satisfied | If the Flight service reaches faulted-state by encountering flight unavailability failure, RETRY will be the next state reached by the Flight service. |
| A<> erratic_flight.F_FAULTED imply f_handler.CALL_REC_ACTION | Satisfied | Failure state reached by a Flight service will eventually be dealt by calling a recovery action at the Fault-handler |
| A<> erratic_flight.RETRY and f_handler.tries>3 imply f_handler.CALL_ALT | Satisfied | When maximum retries for the availability of Flight service have elapsed, Fault-handler calls the alternate service. |
| erratic_flight.S_FAULTED and erratic_flight.fs_fault==SEATS_UNAVAILABLE –> f_handler.CALL_ALT | Satisfied | Seats unavailability failure of Flight service is dealt by calling the alternate service by the Fault-handler. |
| **Hotel Service** | | |
| A[] f_handler.WAIT_H and f_handler.wait>3 imply f_handler.H_RETRY | Satisfied | It is always the case that the Retry action on the Hotel service can only be called when the Fault-handler has already waited for 3 time units. |
| E<> erratic_hotel.H_FAULTED and erratic_hotel.hs_fault==H_UNAVAILABLE imply erratic_hotel.RETRY | Satisfied | Unavailability failure reached by the Hotel service will be followed by a situation where Retry action can be invoked. |
| A<> (erratic_hotel.H_FAULTED and erratic_hotel.hs_fault==H_UNAVAILABLE) imply f_handler.H_RETRY | Satisfied | Unavailability of Hotel service is dealt by calling "Retry" action by the Fault-handler. |
| A<> f_handler.tries<3 imply f_handler.H_RETRY | Satisfied | Until the maximum number of retries reaches, Fault-handler keeps retrying the Hotel service. |
| **Travel Agent Service** | | |
| E<> QoS<5 imply ta_logic.CALL_SP | Satisfied | If the QoS value of overall process is less than 5 units, Shop service is executed. |
| E<> QoS>5 imply ta_logic.SKIP_SP | Satisfied | Shop service is skipped if the QoS value of the overall process is greater than 5 units. |
| A<> client.END and t_agent.READY and ta_logic.READY | Satisfied | Client's request is completed and the Travel agent is ready to receive a new request. |
| **BA Consistent Outcome** | | |
| A<> coord.ENDED and erratic_flight.READY and f_handler.READY and client.END and flightPC.READY and erratic_hotel.READY and hotelPC.READY and (bike.READY or car.READY) and shop.READY and shopPC.READY and comp_dist.READY and attraction.READY | Satisfied | All the parties involved in the joint operation must finish their work. |
| A<> coord.ENDED and erratic_flight.READY and f_handler.READY and client.END and flightPC.READY and erratic_hotel.READY and hotelPC.READY and (bike.READY or car.READY) and shop.READY and shopPC.READY and comp_dist.READY and attraction.READY and flightPC.p_outcome==P_COMMITTED and hotelPC.p_outcome==P_COMMITTED and tcOutcome==TC_COMMITTED | Satisfied | All the parties involved in the joint operation complete their work and reach a consistent outcome with respect to protocol instance they run on. |

task. **Compute-Distance** service computes the distance of Attraction from the Hotel and communicates it to the Travel agent service. As per our considerations, the distance of Attraction from the Hotel is either five or more kilometers. **Car** and **Bike** services are simple services and interact with the Travel-Agent service to contribute towards the completion of common reservation task. Optional **Shop** service interacts with Travel-Agent service and is used to test the quality-of-service attribute of the overall reservation task. The Shop service is skipped from execution if the overall travel reservation task exceeds 5 time units; in the other case, the Shop service is executed. **Train** service works as an alternate of the Flight service if the Flight service encounters a dormant fault.

The complete UPPAAL model of the travel reservation process is provided at [33].

### B. SYSTEM VERIFICATION

Verification of the implementation builds trust in the correctness of concepts developed during the implementation of the framework. The model is verified to check whether the qualitative requirements are satisfied or not. The essential requirements of the system like safety, reachability, and liveness are verified and formulated using UPPAAL query language (a subset of Computation Tree Logic [34]–[35]). The details of each of the identified requirements (properties) are given below:

#### 1) SAFETY PROPERTIES

A safety property checks if a specific condition holds in all the states of the execution paths. Its UPPAAL formulation is: `A [] p`, i.e., "Always globally p", which means that, for all executions paths `p` (property) holds in all the states of those paths. As our specific example, it is always the case that after the maximum number of re-tries elapses fault-handler never calls the retry action for the flight and hotel services.

#### 2) REACHABILITY PROPERTIES

A reachability property amounts to check whether a specific condition holds in some state of the execution path of the system behavior. The UPPAAL formulation of this property is: `E <> p`, meaning "Exits eventually p", stating that there is an execution path in the system behavior such that `p` (property) holds in some state of that execution path. In our specific case, the overall reservation process can reach a state where the QoS value may exceed the constrained time limit.

#### 3) LIVENESS PROPERTIES

A liveness property guarantees that a specific condition (something good) will eventually hold at some point. The UPPAAL formulation of this property is: `A<> p`, i.e., "Always eventually p", meaning that, for every execution path `p` (property) holds at least in one state of each path. Another form of liveness property is "leads to", which is formulated using, `q -> p`, meaning that any execution path starting with a state in which `q` holds, later reaches a state in which `p` also holds. For our reservation scenario, eventually, all roles involved in the reservation task reach their end-states, and the final outcome (committed or aborted) of the overall process is also decided. Table 4 gives UPPAAL formulation, status, and description of each of the verified properties. Moreover, Fig. 13 gives a summary of verification results for Flight, Hotel, and Travel agent services. The verification results show that the fault-handler successfully deals with all considered failure types encountered by the participant services. Furthermore, verification results given in Table 4 also prove that the property of reaching a consistent outcome on the completion of joint task also holds.

| Recovery Strategy | Fault Type: System Faults — Service unavailability | Inconsistent Faults — Seats unavailability | Interaction faults — QoS | Interaction faults — Time-out |
|---|---|---|---|---|
| Wait | ✓ | | | ✓ |
| Retry | ✓ | | | ✓ |
| RetryUntil | ✓ | | | ✓ |
| Alternate | | ✓ | | |
| Skip | | | ✓ | |

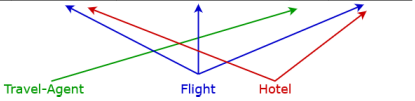


**FIGURE 13.** Results.

## VIII. CONCLUSION AND FUTURE WORK

Web services business activity (WS-BA) protocol is designed to coordinate the actions of composite services to reach a common agreement on the outcome of joint operations. However, in its current settings, the BA protocol deals with failures using compensating actions in backward recovery fashion, which is an expensive and time-consuming process. This paper extends WS-BA protocol with Fault-handler to deal with failures using forward recovery approach. The Fault Fault-handler is implemented as a separate process to be used with varying requirements of different application examples. The Fault-handler implements a pool of recovery actions for each of the considered failure-types affecting the execution of participant services. The study is supported by the implementation of a well-known travel reservation example. The key properties of the system, like safety, reachability, and liveness have been verified using the model-checking and verification tool UPPAAL. Verification results prove that when a failure occurs during the execution of a participant service, its corresponding recovery action is invoked to recover from that failure to proceed further rather than to compensate or perform the operation all-over-again. In addition to that, the property of reaching a consistent agreement on the outcome of joint operation has also been verified. It is concluded that the introduction of a fault-tolerance mechanism in WS-BA protocol helps to detect and resolve failures at runtime to provide reliable services. Due to its basis on the forward

recovery approach, the proposed approach saves time and cost and builds trust to be used in sensitive and mission-critical applications.

### A. FUTURE WORK

This study implements fault-handling actions for the participant role of the BA protocol with which the participant services register for communication with the composite service. However, there is a possibility that the coordinator (or composite) service itself fails. Due to the complexity of the model and the state-space explosion problem of the model-checker UPPAAL, we could not implement recovery actions for the composite (or coordinator) service. However, the model can be simplified to implement recovery solutions for the coordinator role of the protocol. Moreover, apart from different recovery actions and their combinations presented in the paper, few more actions and their combinations can also be suggested. Furthermore, which of the combinations are optimal for resolving the same type of failures and how to obtain them are questions that may be explored in detail.

## REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architectures and Applications*, 1st ed. Berlin, Germany: Springer-Verlag, 2010.

[2] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Inf. Sci.*, vol. 280, pp. 218–238, Oct. 2014.

[3] Gartner. (2017). *Gartner Identifies the Top 10 Strategic Technology Trends for 2018*. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-10-04-gartner-identi_es-the-top-10-strategic-technology-trends-for-2018

[4] D. B. Claro, P. Albers, and J.-K. Hao, "Web service composition," in *Semantic Web Services, Processes and Applications*. Boston, MA, USA: Springer, 2006, ch. 8, pp. 227–245.

[5] E. Newcomer and I. Robinson. (2009). *Web Services Coordination (WS-Coordination) Version 1.2*. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html

[6] E. Newcomer and I. Robinson. (2009). *Web Services Atomic Transaction (WS-Atomic Transaction) Version 1.2*. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec.html

[7] E. Newcomer and I. Robinson. (2009). *Web Services Business Activity (WS-Businessactivity) Version 1.2*. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os.pdf

[8] Q. Wang, G. Lv, S. Ying, and J. Wen, "A policy-driven exception handling approach for service-oriented processes," in *Proc. IEEE 16th Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, May 2012, pp. 49–455.

[9] A. Liu, Q. Li, L. Huang, and M. Xiao, "FACTS: A framework for fault-tolerant composition of transactional web services," *IEEE Trans. Services Comput.*, vol. 3, no. 1, pp. 46–59, Jan. 2010.

[10] C. Liu and X. Zhao, "Towards flexible compensation for business transactions in web service environment," *Service Oriented Comput. Appl.*, vol. 2, nos. 2–3, pp. 79–91, Jul. 2008.

[11] A. P. Ravn, J. Srba, and S. Vighio, "A formal analysis of the web services atomic transaction protocol with UPPAAL," in *Proc. 4th Int. Conf. Leveraging Appl. Formal Methods, Verification, Validation*, 2010, pp. 579–593.

[12] S. Choi, H. Jang, H. Kim, J. Kim, S. M. Kim, J. Song, and Y.-J. Lee, "Maintaining consistency under isolation relaxation of web services transactions," in *Proc. 6th Int. Conf. Web Inf. Syst. Eng.*, in Lecture Notes in Computer Science, vol. 3806. New York, NY, USA: Springer, Nov. 2005, pp. 245–257.

[13] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *Proc. 4th Int. School Formal Methods Design Comput., Commun., Softw. Syst. (SFM-RT)*, vol. 3185. Berlin, Germany: Springer-Verlag, 2004, pp. 200–236.

[14] Gartner. (2017). *Gartner Identifies the Top 10 Strategic Technology Trends for 2018*. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-10-04-gartner-identifies-the-top-10-strategic-technology-trends-for-2018

[15] J. Steiman. (2018). *Six Trends That Will Shape Customer Service in 2018 (for Better or Worse)*. [Online]. Available: https://www.forbes.com/sites/theyec/2018/02/08/six-trends-that-will-shape-customer-service-in-2018-for-better-or-worse/#6ddf7246672a

[16] D. Petrova-Antonova, D. Manova, and S. Ilieva, "Testing web service compositions: Approaches, methodology and automation," *Adv. Sci., Technol. Eng. Syst. J.*, vol. 5, no. 1, pp. 159–168, Jan. 2020.

[17] J. Zhang, A. Zhou, Q. Sun, S. Wang, and F. Yang, "Overview on fault tolerance strategies of composite service in service computing," *J. Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–8, Jun. 2018.

[18] N. Memon, M. S. Vighio, and Z. Hussain, "Web services failures and recovery strategies: A review," *Indian J. Sci. Technol.*, vol. 12, no. 43, pp. 1–6, Nov. 2019.

[19] P. P. W. Chan, M. R. Lyu, and M. Malek, "ReliableWeb services: Methodology, experiment and modeling," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Salt Lake City, UT, USA, Jul. 2007, pp. 679–686, doi: 10.1109/ICWS.2007.151.

[20] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[21] R. Angarita, Y. Cardinale, and M. Rukoz, "Reliable composite web services execution: Towards a dynamic recovery decision," *Electron. Notes Theor. Comput. Sci.*, vol. 302, pp. 5–28, Feb. 2014.

[22] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy, "Dependability in the web services architecture," in *Architecting Dependable Systems* (Lecture Notes in Computer Science), vol. 2677, R. de Lemos, C. Gacek, and A. Romanovsky, Eds. Berlin, Germany: Springer, 2003, pp. 90–109.

[23] M. Little, "Transaction and web services," *ACM J. Commun., Assoc. Comput. Machinery*, vol. 46, no. 10, pp. 49–54, 2003.

[24] G. Pardon. *Business Transactions, Compensation and the Try-Cancel/Confirm (TCC) Approach for Web Services*. Accessed: Jan. 5, 2021. [Online]. Available: https://cdn.ttgtmedia.com/searchWebServices/downloads/Business_Activities.pdf

[25] Z. Yang and C. Liu, "Implementing a flexible compensation mechanism for business processes in Web service environment," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Chicago, IL, USA, Sep. 2006, pp. 753–760.

[26] M. M. Schäfer and P. W. D. Nejdl, "An environment for flexible advanced compensations of web service transactions (ICWS'06)," *ACM Trans. Web*, vol. 2, no. 2, pp. 1–14, 2008.

[27] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz, "Measuring fuzzy atomicity for composite service execution," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Aug. 2016, pp. 26–71.

[28] L. Zeng, H. Lei, J.-J. Jeng, J.-Y. Chung, and B. Benatallah, "Policy-driven exception-management for composite web services," in *Proc. 7th IEEE Int. Conf. E-Commerce Technol. (CEC)*, Jul. 2005, pp. 355–363.

[29] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[30] A. Liu, Q. Li, and M. Xiao, "A declarative approach to enhancing the reliability of BPEL processes," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2007, pp. 272–279.

[31] K. S. M. Chan, J. Bishop, J. Steyn, L. Baresi, and S. Guinea, "A fault taxonomy for web service composition," in *Proc. Int. Conf. Service-Oriented Comput. (ICSOC)*, 2009, pp. 363–375.

[32] R. Hamadi, B. Benatallah, and B. Medjahed, "Self-adapting recovery nets for policy-driven exception handling in business processes," *Distrib. Parallel Databases*, vol. 23, no. 1, pp. 1–44, Feb. 2008.

[33] S. Vighio. *UPPAAL: Travel Reservation Process (UPPAAL Models)*. Accessed: Feb. 9, 2021. [Online]. Available: https://sites.google.com/a/quest.edu.pk/vighio/home/fault-tolerance-model

[34] S. Wimmer and P. Lammich, "Verified model checking of timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems* (Lecture Notes in Computer Science), vol. 10805, D. Beyer and M. Huisman, Eds. Cham, Switzerland: Springer, 2018, pp. 61–78.

[35] A. P. Ravn, J. Srba, and S. Vighio, "Modelling and verification of Web services business activity protocol," in *Tools and Algorithms for the Construction and Analysis of Systems* (Lecture Notes in Computer Science), vol. 6605, P. A. Abdulla and K. R. M. Leino, Eds. Berlin, Germany: Springer-Verlag, 2011, pp. 357–371.

**NIMRA MEMON** received the bachelor's degree in computer science and the master's degree in information technology from the Quaid-e-Awam University of Engineering, Science and Technology, Nawabshah, Pakistan. She is currently pursuing the Ph.D. degree in information technology. She is currently working as a Lecturer with the Department of Computer Science, Government Girls Degree College Nawabshah, Nawabshah.

**MUHAMMAD SALEEM VIGHIO** received the M.Sc. and Ph.D. degrees in computer science from Aalborg University, Denmark, in 2009 and 2012, respectively. He is currently working as an Associate Professor and the Head of the Department of Computer Science, Quaid-e-Awam University of Engineering, Science and Technology, Nawabshah, Pakistan. Since 2012, he has been produced several master's and Ph.D. students. He has been credited with several national and international conference papers and journal articles. His research work focuses on the verification of software systems, including real-time and embedded systems, and web services protocols. He is a member of the editorial boards of many research journals. He is also a member of the board of studies of national universities.

**YAR MUHAMMAD** (Senior Member, IEEE) received the master's degree in computer engineering from Mid Sweden University, Sweden, in 2009, and the Ph.D. degree in information communication technology from the Tallinn University of Technology, Estonia, in 2015. He has taught with the University of Tartu, Estonia. He is currently working as a Senior Lecturer (Assistant Professor) with Teesside University, U.K. He received a Young Investigator Award, which was awarded by Springer and IFMBE at 16th Nordic-Baltic Conference on Biomedical Engineering & Medical Physics and Medicinteknikdagarna 2014, Sweden; and he was runner-up for the Best Paper Award in the 26th ISSC, Ireland, in 2015.

**ZAHID HUSSAIN ABRO** received the B.Sc. and M.Sc. degrees in computer science from the University of Sindh, Jamshoro, Pakistan, and the Ph.D. degree in computer science from the Graz University of Technology, Austria, in July 2010. More than 20 M.S. students and three Ph.D. students have completed their studies under his supervision. He has organized four national conferences. His research interests include software engineering, particularly agile software development methods, HCI, mobile HCI, user experience, agile user experience, mobile learning, and web engineering. He was a member of Federation of Pakistan, Chambers of Commerce and Industry's Standing Committee on Research and Development (Policy) for the years 2017 and 2018 at the national level.